Term Project: (This term project will be divided into several modules (i.e., assignments). We will use what we have learned from our lecture notes to design and implement an intelligent agent. We are not interested in using any existing implemented maze problem solver.

Project description:

Consider the problem of developing (designing and implementing) an intelligent agent called the maze problem-solver (or called a mouse).

Given ANY maze configuration, the maze has an entrance (equivalently, called the initial state) for entering the maze, and two exits, g1, and g2 (or called final states). The maze problem-solver can find its way from an entrance **[ ]** to an exit **] [.** The goal is that the maze problem-solver enters the given maze and finds a way to exit the maze with a minimal time span (in terms of the number of squares traversed).

A maze configuration is given at the end of this assignment. We will continue to use this maze configuration for the remaining assignments (i.e., project). This project is to program an agent, called the maze problem-solver, which will run around any given maze to find its way out from an exit. We will address the maze configuration later and its representation, which could serve as input to the program.

**Percept and State Representations**

An *agent program* can be defined as an agent function, *f*: P\* $\rightarrow $ A, which maps every *possible* percept sequence to *a possible* action the agent can perform.

Given a maze, let the wall line have the value 1, the grid line is 2, and the entrance/exit line with a hole is 0. For each grid, we will use these values to describe the four sides of any grid, from the top (the head of the agent (mouse)) *h*, then its right *r*, its rear (its tail) *t*, and then its left *l*. That means, at any grid, there corresponds a percept (perceived by an agent) that can be represented as In(h, r, t, l) which forms a state [*h, r, t, l*], where 0 $\leq $ *h, r, t, l* $\leq $ 2. This also coincides with the fact that the agent is heading to a side h of a grid and the rear of the agent is pointing to the side t, opposite to the side h, of the grid square. Then, the right and left sides of the agent are the r and l sides. The specification of a percept and its state is a one-to-one correspondence. For example, if the agent (mouse) enters grid A, the percept for the grid can be represented as In(A), which is In(2, 2, 0, 1). The state’s specification of the percept In(2, 2, 0, 1) is [2, 2, 0, 1]. Therefore, it suffices to use the state’s specification [*t, r, b, l*] to represent the percept In(h, r, t, l). For this reason, we will use the state’s specification to represent states and percepts. If we use In(A) as an initial state, then the state can be represented as [2, 2, 0, 1]. However, if we insist the agent should be outside the maze facing the entrance, then the initial state [0, -, -, -] or [0, 2, 2, 2] where the precept is In(0, -, -, -) or In(0, 2, 2, 2), respectively. All these descriptions are conformed with the Simple-Reflex-Agent(percept). It states

state $\leftarrow $ INTERPRET\_INPUT(percept)

where percept is the physical configuration of the grid generated by the sensor of the agent. What it means is that the image is converted into a program variable, state, using the function INTERPRET\_INPUT.

The agent (the mouse) has three actions, namely, rightTurn, leftTurn, and forward.

The rightTurn (R) with respect to a grid is defined as follows:

**rightTurn**(In(2, 2, 0, 1), In(2, 0, 1, 2)). Brings the first tuple to the last.

In fact “In” can be omitted. Therefore, we write

**rightTurn**([2, 2, 0, 1], [2, 0, 1, 2]).

This means, that the agent (mouse) is first in a grid heading in a direction of an open side of the grid. The rightTurn action enables the agent to remain in the same grid but the agent turns right heading in a new direction that is perpendicular to the initial direction, facing an open side of the grid. Pictorially,

 [2, 2, 0, 1] [2, 0, 1, 2].

Using the state’s specifications, it is **rightTurn**([2, 2, 0, 1], [2, 0, 1, 2]), or

 R

[2, 2, 0, 1] $\rightarrow $ [2, 0, 1, 2].

The leftTurn(L) with respect to a grid is defined as follows:

**leftTurn**(In(2, 2, 0, 1), In(1, 2, 2, 0). Brings the last tuple to become the first tuple.

This means the leftTurn action enables the agent to remain in the same grid but the agent is in a new direction that is perpendicular to the initial direction and faces to the left side of the grid of its initial direction. Pictorially,

 2 2

 1 2 1 2

 0 0

 [2, 2, 0, 1] [1, 2, 2, 0]

Using the state’s specifications, it is **leftTurn**([2, 2, 0, 1], [1, 2, 2, 0], or

 L

[2, 2, 0, 1] $\rightarrow $ [1, 2, 2, 0]

The forward (F) with respect to adjacent grids is defined as follows:

For the following pictorial example, we can describe the agent going forward from one grip to another grip with different percepts perceived by the agent. This first four-tuple percept describes that the agent (mouse) is in a grid and then runs into the other grid with two walls on its right and left sides, and two grid lines on its front and rear sides. We use forward action to describe the following:

**forward**(In(2, 2, 0, 1), In(2, 1, 2, 1))

 2 2

 1 1 1 1

2

2

 1 2 1 2

 0 0

Using the state’s specifications, it is **forward**([2, 2, 0, 1], [2, 1, 2, 1]), or

 F

[2, 2, 0, 1] $\rightarrow $ [2, 1, 2, 1]

**Distinct Configuration Grids**

**Using the above description as an approach, give/describe your proposed percepts and actions.** (Warning: the later assignments will be based on your previous work! and therefore, good detailed work would help you with the rest of your entire project (i.e., assignments).

1. How many different configurations of grids are in the given maze?
2. How many different actions are based on the given maze? Specify each of them.

There are seven distinct configurations of grids from the given maze.

1st

 [2, 2, 0, 1]

2nd

 [1, 2, 2, 2] [2, 2, 1, 2] [2, 2, 2, 1] [2, 1, 2, 2]

3rd

 [2, 1, 2, 1] [1, 2, 1, 2]

4th

 [1, 2, 2, 1] [2, 2, 1, 1] [1, 1, 2, 2] [2, 1, 1, 2]

5th

 [2, 1, 1, 1] [1, 1, 2, 1] [1, 2, 1, 1] [1, 1, 1, 2]

6th

 [2, 2, 2, 2]

7th

 [0, 1, 2, 1] [2, 1, 0, 1]

 g1 g2

**States Space Specification**

Given the 1st configuration of the grid, the state associated with this node would be:

 state

Given the 2nd configuration of the grids, the state associated with this node would be:

Among the grids of the 2nd configuration of the grids, they are described by the inner states of the state.

Given the 3rd configuration of the grids, the state associated with this node would be:

Given the 4th configuration of the grids, the state associated with this node would be:

Given the 5th configuration of the grids, the state associated with this node would be:

Given the 6th configuration of the grids, the state associated with this node would be:

Given the 7th configuration of the grids, the state associated with this node would be:

Let’s normalize the standard rule for specifying the inner four states. Given a maze, we specify a state for a perceived percept beginning with [*h, r, t, l*], disregarding the direction of the agent.

 h

 l r

 t

**Goal Test**

For the agent to reach the goal grid, the obtained resultant state must be equal to an inner state [0, 1, 2, 1] and meet the requirement of the minimal cost g(goal-node), which is the total number of turns and the total number of grids the agent goes forward.

**Performance measure**

To measure the cost g(goal-node) which is obtained from the entrance grid (the root of the state space tree search graph) to an exit grid (goal node). The solution would be the path from the root to the goal node. Could we use the goal-based model to implement the agent, the maze problem-solver? (We will continue to develop this section.)

We also can add a variable nosTurns to count the number of turns the agent did in a grid, and another variable nosGridsCovered to count the number of forwards the agent executed between two labeled grids (nodes).

**Q1:** [40 points]

Convert the given maze into a graph. (Hint: For simplicity’s sake, use those specified grids (with the alphabet) to name the nodes, and their distance is the number of grids in between the paired named nodes as their distance. For example, node C and node R are reachable by forwarding 6 grids from C towards R, or from R towards C.) We give labels for 55 out of 400 grids. That is, there are 55 labeled nodes in the graph. The total number of grids labeled is minimal. This ensures that the generated connected undirected weighted graph is one-to-one correspondence to the given maze’s configuration.

Given the graph and a maze configuration, for each labeled node of the graph, we need to construct its associated state; and for each edge between two labeled nodes, if these nodes are next to each other without another grid in between them, then specify a forward.Action associated with this edge, or if these nodes are not next to each other, but there is an avenue of grids between these two labeled nodes, then specify a state.Action associated with this edge. An example is given at the end of this description.

A state.Action between two labeled nodes (labeled grids) is a description of a sequence of actions taken in grids, where the current action may yield another percept. If the edge between two nodes represents a sequence of grids and thus a sequence of states for these grids. The edge between two labeled nodes is associated with state.Action.

END OF PROBLEM.

**The following graph is a maze example: Assume A is the entrance [ ]. Both g1 and g2 are exits ] [.**

**] [**

**] [**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **C** |  |  |  |  |  | **R** |  |  |  |  |  |  | **k** |  |  |  |  | **w** | **g1** |
|  |  |  |  |  |  | **Q** |  |  | **b** |  |  |  | **j** |  |  |  |  |  |  |
| **B** |  |  |  |  | **O** |  |  |  |  |  |  |  | **i** |  |  |  | **r1** |  | **z** |
|  |  |  |  | **L** | **N** |  |  |  |  | **c** |  | **e** |  |  |  | **o** | **r** | **v** |  |
|  |  | **E** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | **a** |  |  |  | **h** |  |  |  |  |  |  |
|  |  | **D** | **H** | **K** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | **P** |  | **Y** |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | **J** | **M** |  | **T** |  |  |  |  |  |  |  |  |  | **q** | **u** | **y** |
|  |  |  | **G** | **I** |  |  |  |  | **Z** |  |  |  |  |  |  | **n** | **p** | **t** | **x** |
|  |  |  |  |  |  |  |  | **X** |  |  |  | **d** | **g** |  |  |  |  |  |  |
|  |  |  |  |  |  |  | **S** | **V** |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **m** |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  | **f** |  |  |  |  |  |  |
| **A** |  |  | **F** |  |  |  |  | **U** |  |  |  |  |  |  | **l** |  |  | **s** | g2 |

**[ ]**

**The following description could help you to get Q1 done.**

An agent’s position in a grid can be specified as {h, r, t, l| 0 $\leq h, r, t, l \leq $ 2; 0 is a hole; 1 is a wall, and 2 is open (a grid line)}. The percept of a grid perceived by an agent can be normalized as [h, r, t, l], where the first parameter is the side of the grid where the agent heading; the second parameter is the side of the grid at the right-hand side of the agent, the third parameter is the side of the grid at the rear (tail) of the agent, and the fourth parameter is the side of the grid at the left-hand side of the agent. Pictorially, we have

 2 2

 1 2 1 2

 0 0

 [2, 2, 0, 1] [1, 2, 2, 0]

 leftTurn([2, 2, 0, 1], [1, 2, 2, 0])

 [2, 2, 0, 1] [2, 0, 1, 2]

 rightTurn([2, 2, 0, 1], [2, 0, 1, 2])

 2 2

 1 1 1 1

2

2

 1 2 1 2

 0 0

 forward([2, 2, 0, 1], [2, 1, 1, 1])

Without loss of generality, we always use [*h, r, t, l*] to define the state of a percept. A state has four components. Each of the components of a state describes the agent being in a grid and where it is heading to.

The stateAction for the avenue between node A and another labeled node

F

A

RT

[2, 1, 2, 1]

[1, 2, 1, 2]

LT

LT

RT

LT

The stateAction for the avenue between node A and another labeled node

RT

RT

[1, 2, 1, 2]

[2, 1, 2, 1]

F

F

LT

F

F

F..

RT

[1, 2, 1, 2]

[2, 1, 2, 1]

LT

LT

RT

LT

RT

RT

RT

[2, 0, 1, 2]

[2, 2, 0, 1]

[1, 2, 1, 2]

[2, 1, 2, ])

LT

LT

RT

LT

LT

RT

F

RT

[0, 1, 2, 2]

[1, 2, 2, 0]

LT

state

The stateActon for the avenue between node A and another node

node

The state for node A

For each node, there corresponds to a state of the percept perceived by the agent, which is as follows:

 a

 d b

 c

we will have four components that uniformly specify in the order that begins with [a, b, c, d], [b, c, d, a], [c, d, a, b], and [d, a, b, c], where 0 $\leq a, b, c, d\leq $ 2; 0 is a hole; 1 is a wall, and 2 is open (a grid line). Each of the components characterizes the position/direction of the agent. If the agent is heading toward a, then the description of the state would be [a, b, c, d]. If the agent is heading toward b, then the description of the state would be [b, c, d, a]. If the agent is heading toward c, then the description of the state would be [c, d, a, b]. If the agent is heading toward d, then the description of the state would be [d, a, b, c]. Pictorially,

 a a

 d b d b

 c c

the state of this percept is [a, b, c, d], and the state of this percept is [b, c, d, a].

Given a maze, it can construct a 1-1 corresponding graph. From the graph with the given maze configuration, we can construct a state space graph. For each node of the given graph, there corresponds to a state of four percepts. The state characterizes each of the four percepts perceived by the agent in different positions to form a four-tuple state, which is of the form [*h, r, t, l*]. The are four four-tuple states within a state. Each of these four-tuple states describes the direction of the agent in the grid. Within a grid, the agent can move forward to another grid, and right turns or left turns within the grid. The following is a partial **state space graph**. In it, a node corresponds to a state, an edge corresponds to action (if the node is adjacent(next) to the other node), and an edge corresponds to a state action (if the node is connected by an edge that represents more than one grid in between.

stateAction

state

F1

F1

F1

F1

 R

[2, 2, 1, 2] [2, 1, 2, 2]

 L R L L L R

[2, 2, 2, 1] [1, 2, 2, 2]

 R

state

 R

[1, 2, 1, 2] [2, 1, 2, 1]

 L R L L L R

[2, 1, 2, 1] [1, 2, 1, 2]

 R

 R

[2, 2, 0, 1] [2, 0, 1, 2]

 L R L L L R

[1, 2, 2, 0] [0, 1, 2, 2]

 R

stateAction

F6

 R

[2, 1, 2, 1] [1, 2, 1, 2]

 L R L L L R

[1, 2, 1, 2] [2, 1, 2, 1]

 R

F1

F1

F1

F6

F1

 R

[2, 2, 2, 1] [2, 2, 1, 2]

 L R L L L R

[1, 2, 2, 2] [2, 1, 2, 2]

 R

 state

F1

 R

[2, 1, 2, 1] [1, 2, 1, 2]

 L R L L L R

[1, 2, 1, 2] [2, 1, 2, 1]

 R

 stateAction

**F1**

F1

F1

 R

[1, 1, 1, 1] [1, 1, 1, 1]

 L R L L L R

[1, 1, 1, 1] [1, 1, 1, 1]

 R

state

Action F

F1

F1

F1

F1

state

Action F

 R

[2, 1, 2, 2] [1, 2, 2, 2]

 L R L L L R

[2, 2, 1, 2] [2, 2, 2, 1]

 R

 R

[1, 2, 2, 2] [2, 2, 2, 1]

 L R L L L R

[2, 1, 2, 2] [2, 2, 1, 2]

 R

stateAction

F1

state

F2

 R

[2, 1, 2, 1] [1, 2, 1, 2]

 L R L L L R

[1, 2, 1, 2] [2, 1, 2, 1]

 R

state

 R

[1, 2, 2, 2] [2, 2, 2, 1]

 L R L L L R

[2, 1, 2, 2] [2, 2, 1, 2]

 R

 R

[1, 1, 2, 2] [1, 2, 2, 1]

 L R L L L R

[2, 1, 1, 2] [2, 2, 1, 1]

 R

F1

F1

F2

F1

Notes:

This is a state-space graph.

Algorithms traverse the graph from one node to another node, which will be verified by state space (which is the percepts of the environment of grids).

Algorithms traverse the graph to generate a state space tree graph.

Input is a graph with state space. (adjacent list of the graph).

Future Research (I will discuss the following in class and will take time to develop its description):

Develop a theory for claiming annotations, A, B, …, Z, a, b, …, z, which are necessary and sufficient. Obtain a connected undirected weighted graph from a given maze. Will this obtained graph uniquely correspond to the given maze?

Design and implement the agent (the maze problem-solver, a mouse, a program, or a simulator), such that for given any maze, the agent can find an exit efficiently, once it enters the maze.

 Input to the agent is any maze. (the agent can take different mazes as its input).

 The agent consisting algorithms that traverse a given maze to find an exit.

 Tree search algorithms for finding the solution, which is a path from the root node to the goal node.